Towards a low-delay Internet

Niels Möller

2012

Available at http://www.lysator.liu.se/~nisse/network-book/.

Preface

The goal of this book is to describe two ideas which I believe can improve any Internet application which is sensitive to latency, and can do so in a reasonably minimalistic way, without introducing much new complexity. Nonetheless, even a minor upgrade of the Internet infrastructure is a daunting task... It is my hope that this book can help prod the evolution of the Internet a bit in the right direction.

The first idea, covered in Chapter 2, is a minimalistic "quality-of-service" mechanism which lets applications in end noods chooose if they prefer packet losses or delays. This is not my idea, I just believe it's pretty important.

The second idea is way to use a small amount of additional signalling (similar to current explicit congestion notification, ECN), together with a new congestion control algorithm. The main difference from methods based on additive increase and multiplicative decrease is that we do smaller and more frequent updates to the sender's window size. With no large changes to the window sizes, we can reduce both average queue size and queue variations in bottleneck routers, which in turn means reduced queueing delays. This is described in Chapter 4, with preceding chapters giving some of the needed background.

I would like to encourage others to experiment further with these ideas. To help you get started, Chapter 5 provides draft protocol specification, suitable for testing on a lab network.

Along the way, I hope to convey some of the insights in window-based congestion control which I gained during my PhD work at the automatic control department at KTH. I'll do my best to describe the main ideas without resorting to long mathematical formulas, but I collect some of the related math to appendices for mathematically inclined readers.

This is work-in-progress

This book is still has many rough spots, since it's partly a mashup of texts from different publications. Some of the more important issues remaining are:

Chapter 1: Try to simplify the description of objectives and constraints; it's mostly copied from the thesis. Discuss the challenges to control (uncertainty, scarce signalling, scalability). Maybe also summarize wireless issues. Any other background material needed?

Chapter 3: Needs a block diagram and some step response figures.

Chapter 4: Include some analysis of equlibrium and fairness properties. Review evaluation section, it's copied from earlier article drafts. It would be interesting to include the more recent "Binary increase congestion control" (BIC) in the comparison.

References: Not all results and ideas described in these pages are my own. I currently do a poor job of pointing to sources and give proper credit for research.

Various smaller things are noted with "FIXME" comments throughout the text.

Contents

The Internet

CONGESTION control is one piece of the Internet machinery. But how does it fit in the larger picture, and how does the Internet really work? A younger second cousin of mine, Tom, once asked me how life was before the Internet. Let us meet again in a few years, and continue that conversation.

History

TOM: Could you use the Internet when you went to school?

NIELS: No, I got my first account an a machine with a real Internet connection in October 1992, two years year after I entered university.

TOM: So that was the first time you used email?

NIELS: Not quite, I got a student account at the CS department one and a half year earlier. The computer system for the students was not directly connected to the Internet, presumably for the protection of the Internet, but one could send and receive email, both locally, to other students and teachers, and to the outside world.

Since email was the first big application for the network, it was common with gateways between Internet email, and other systems. I remember having a summer job in the early 1990s at a telecommunications company. We had email, but no direct connectivity to the Internet. It was not possible to connect directly to FTP servers around the world, but some file archives offered an FTP by email service, which we used.

TOM: So how did people exchange files before the Internet?

NIELS: Computer communication was fairly obscure; some people used modems to dial in to bulletin board systems, and hacker groups would exchange data on floppy disks sent via postal mail. It was common that computer magazines and books published program listings, but to try the program, you had to first type it into your computer by hand.

TOM: When did Sweden get connected to the Internet?

NIELS: I think the first network that was connected was the university network SUNET. A 56 kbit/s satellite link was set up between the Nordic

university networks, and the John von Neumann Center at Princeton, New Jersey. Due to a delay on the American side of the link, Sweden missed the outbreak of the first Internet worm, released by Robert Morris on November 2, 1988[1]. The satellite link was operational a week later[2].

TOM: It's hard imagine how it was before the Internet, I use email and instant messaging constantly to keep in touch with friends.

NIELS: The phone system is of course many decades older than the Internet, so that's what people were using to keep in touch. Today, the Internet is used for everything, not just email and file transfer, but phone calls, radio, film distribution, etc. Before this convergence, we had a couple of large but separate communication systems, starting with radio broadcast and the (wired) telephone network, then television (different systems for terrestrial, satellite and cable), and finally mobile telephone networks.

Circuit switching vs packet switching

TOM: How does the Internet work, then? What makes the Internet different from the telephone network?

NIELS: The telephone systems were traditionally circuit switched. Originally, when making a phone call, you had an operator manually patching together a physical electrical circuit between the two telephones. Later on, this manual patching was replaced by automatic electro-mechanical switches, reacting to each digit of the phone number as you dialled, and then by digital systems.

TOM: This seems easy enough for local calls; there's a cord from each telephone to a huge switch board at the telephone switch, and to connect two telephones, you patch the corresponding two cords together. But how could you patch together a long distance call?

NIELS: For the duration of a call, you need a path reserved through the network, from one switch to the next. Between neighboring switches, your call might use a separate cord, a frequency band in a cable using frequency division multiplex, or certain time slots in digital system with time division. This means that every switch on the path must be aware of your call, and change their state when the call is set up or torn down.

TOM: I've heard that the Internet is "packet switched", what does that mean?

NIELS: Too see the difference between circuit switching and packet switching, say you want to transport fuel from the harbor at Värtahamnen to Arlanda airport. One alternative is to construct a pipeline between these

[1]The worm infected VAX computers and Sun workstations. It is guessed to have infected 6 000 out of the roughly 60 000 computers connected to the Internet at the time.

[2]Kaarina Lehtisalo, *The History of* NORDU*net—Twenty-five years of networking cooperation in the Nordic countries*, `http://www.nordu.net/history/book.html`.

two points. A different way to solve the problem is to load the fuel onto trucks, and let each truck find its way from the harbor to the airport independently. Pipelines are sure useful in certain circumstances, but using a general purpose road network is more flexible. You can freely mix vehicles of different sizes and with different cargo, while you can't use the same pipeline system to transport aviation fuel and beer.

Routing

TOM: But unlike trucks, packets have no drivers?

NIELS: There have actually been some research on "active networking", where each packet contains the intelligence needed for it to find its way through the network[3]. But in the Internet, packets are not smart enough to find their way by themselves.

TOM: So when I send a packet over the network, how does the network know where to send it?

NIELS: Each packet includes its source and destination address. These are the IP-addresses, 32 bits, usually written as, e.g., `130.236.254.103`[4]. Have you ever manually configured the IP address of your computer?

TOM: I think I had to do that sometimes, but that was many years ago, so I don't quite remember. Nowadays it's always automatic. But I remember having to type in some numbers for the address, and also for something called "netmask" and "default gateway", whatever that meant.

NIELS: That's the first step of routing. Your computer needs to know which addresses belong to computers on the same local network as you, and that's what the netmask is; it gives the size of the prefix identifying your local network. On the Lysator network, the machine where I usually read email has the address `130.236.254.103`, and the netmask is `255.255.255.0`. That tells the computer that all addresses starting with `130.236.254.` belong to computers on the same local network. This is the network prefix, and in this case, it's 24 bits long.

When I send a packet, the IP stack first uses the netmask to check if the destination address is on the same local network. If it is, the packet can be sent to the destination directly, without passing through any intermediate routers.

TOM: So that's what the netmask is for. And the "default gateway"?

NIELS: That's the IP address of a router, which must be located on the local network, which routes packets to and from the outside world. All packets sent to destinations beyond your local network, are sent to the default gateway, which then passes them on to other routers. It's called "default",

[3]This goes back at least to the Softnet project in the early 1980s, developed by Jens Zander and Robert Forchheimer at Linköping University.

[4]For IP version 6, the addresses are 128 bits.

because it is possible to set up more complex routing rules, and the default gateway is used when no other rule match.

TOM: So how does the gateway know what to do with my packets?

NIELS: The routing system is more or less automatic. Each router has a couple of in- and outgoing links. It keeps track of which parts of the network are connected, directly or indirectly, to each link. Within an organization, routes can be configured manually, or via protocols such as OSPF (Open Shortest Path First), which exchanges routing information with neighboring routers. The OSPF protocol finds the shortest, or lowest cost, path between each pair of routers. Between organizations, routing depends not only on network topology, but also on private service agreements between Internet Service Providers (ISPs), basically, on who is paying whom. This routing information is exchanged between routers, taking policy into account, using BGP (Border Gateway Protocol).

No matter which protocol or method is used to configure the routing, the end result is a *routing table*, a large lists of network prefixes, and for each prefix, the outgoing link and neighbor router for that prefix. The destination address of each incoming packet is looked up in the routing table to find the longest matching prefix, and then the packet is transmitted to the corresponding router.

Routers close to the network edge often have a default gateway; a neighboring router which is more central, and which is used for all packets not matching any other rule. Routers in the core network don't have any default gateway, and this core is sometimes called the "default-free zone".

TCP/IP

TOM: When people talk about TCP/IP, what does that mean? Is it just a more complicated way to say "Internet"?

NIELS: Well, it's more or less the same thing. At least IP, which simply stands for the Internet Protocol.

If packet switching is analogous to a general purpose road network, then the IP protocol is the standardized freight container for intermodal transport. You pack some goods in a container. Your container is picked up by a truck, then loaded onto a train to a container terminal, then loaded onto a container ship, unloaded at some distant harbor, loaded onto another truck or train, and so on. All without any handling of the cargo inside the container. Before 1970, container transport suffered from several national or company-specific standards, all incompatible with each other. Intermodal freight transport took off after the ISO standardization of container sizes around 1970, transforming the world economy in unexpected ways[5]. **(FIXME: Say something more about the long time needed**

[5]Marc Levinson, *The Box—How the shipping container made the world smaller and the world economy bigger*.

for the rest of society to catch up and take advantage of the new communication possibilities.)

TOM: So there were other packet switching protocols before IP?

NIELS: Sure. I'm not very familiar with any of them, but there were many proprietary networking protocols, e.g., SNA from IBM and DECnet from Digital Equipment Corporation. The x.25 protocol was standardized by CCITT (Comité Consultatif International Téléphonique et Télégraphique) in 1976, primarily for use in networks run by telephone companies. TCP and IP were developed during the 1970s, by the Defense Advanced Research Projects Agency (DARPA) in the United States, as communication protocols for interconnecting different networks. ISO standards for networking were developed during the 1980s, accepting x.25, but not TCP/IP, as a part of the Open Systems Interconnection (OSI) standard. **(FIXME: Were IPX and Apple-talk later or earlier then IP?)**

TOM: But TCP/IP is still not an ISO standard? So what happened with that?

NIELS: When NORDUnet were implementing wide area networking in the late 1980s, the only part of OSI networking standards that was in real use was x.25. And x.25 had both technical and economical problems[6]. So the choice was between the OSI standards, which were not mature, various proprietary networking technologies, and TCP/IP[7]. I imagine the considerations were similar at other organizations planning or building networks.

The first version of NORDUnet was a wide area Ethernet that supported all of IP, DECnet and x.25. Some year later, the network was upgraded to a pure IP network, but still with support for DECnet and x.25 on top of IP.

Architecture

TOM: So IP won the protocol war. Was that just due to good timing, or what is it that makes IP different?

NIELS: As the name implies, IP was designed for the interconnection of heterogeneous networks. I think it managed to find the right, close to minimal, interfaces for doing that. Transmission of IP packets is straightforward to implement on top of virtually any link technology[8]. And on top of IP, end points can implement more sophisticated communication and application protocols. The Transmission Control Protocol (TCP) provides a

[6]Quoting Hans Wallberg, manager of SUNET, "x.25 became terribly expensive when usage grew, since the cost was based on the amount of data transferred. The performance was also too poor. Even if you had a 64 kbit/s connection you never got more than 2400 bit/s due to all overhead."

[7]The so called "protocol wars" are described in some more detail in Kaarina Lehtisalo's book, which is one of my main sources for the history of the early years.

[8]An experimental specification for the transmission of IP packets with carrier pigeons, RFC 1149, was published on April Fool's day 1990. A decade later, this specification was implemented for the first time, in a cooperation between the Linux and BSD User Group in Bergen, and Vesta Brevdueforening.

bidirectional reliable connection between two end points. Most application protocols you use are specified as working on top of any bidirectional data stream, and are used on top of TCP. Some that don't need a connection, or don't need reliable transmission, can work with IP directly[9].

TOM: Isn't that the obvious way to design a networking protocol?

NIELS: It may seem obvious today. But for earlier networks, the applications and the link technologies were more tightly coupled. E.g., in the telephone system, all links and connections were of fixed bandwidth (around 4 kHz for analog transmission and 60 kbit for digital transmission), chosen for supporting a voice service of reasonable quality. And there were few applications besides voice.

TOM: But now your comparing computer network to the telephone system that is many decades older. That's cheating. What if you compare IP to DECnet or to the OSI protocols?

NIELS: I don't know many details about how DECnet works, but one important difference is that IP is an open, non-proprietary standard. You can buy devices that speak IP, network equipment, computers, various gadgets, from a large number of different companies, and they will work together. As for OSI, those protocols were complex committee products, where everybody's favorite feature was included. For example, the OSI model specifies two more layers than in TCP/IP, and the OSI network layer supports both datagram-oriented and connection-oriented services. In TCP/IP, the split between IP and TCP means that the network layer need not be aware of connections. The abstraction of a connection between two hosts is created by the TCP-implementation in the end points. This is an example of the end-to-end principle.

TOM: The "end-to-end principle"? What in the world is that?

NIELS: The end-to-end principle states that in a communications system, as much as possible of the protocol logic and state should be located in the communication end points. This is important for the scalability of the system, and one of the most important principles for the architecture of the Internet. In practice, one direct consequence is that a TCP connection can survive a reboot of routers along the path, as well as routing changes.

TOM: Quite different from the old telephone system.

NIELS: In the telephone system, the telephone devices are simple, and all the complexity is in the network. In the Internet, the network layer is stupid, and instead the end points have to be quite complex[10].

[9] Actually, these protocols usually don't work with IP directly, but with the UDP protocol, which is a very thin layer on top of the best effort IP packet delivery service.

[10] But not as complex as one might think. Adam Dunkel's TCP stack `uip` supports 8-bit micro controllers, and needs about 5 Kbyte for code and a few hundred bytes of RAM, depending on the application.

Congestion control

TOM: When driving, I often get stuck in the traffic, waiting and waiting in some queue. Do packets get stuck too, on the Internet?

NIELS: That's congestion; when there is more traffic than the network can handle. In the Internet, each router has some incoming and some outgoing links of limited capacity. For example, if a router has two incoming links where 80 Mbit of packets arrive every second, and all packets are routed to the same outgoing link, with a capacity of only 100 Mbit/s, then that router is overloaded. Routers buffer packets at the outgoing link, so when the router is overloaded, packets are queued up in that buffer. But unlike traffic queues, the buffer size, and hence the queue size, is limited. When the limit is reached, arriving packets are not added to the queue, they are thrown away or "dropped onto the floor".

TOM: Not quite like road traffic then.

NIELS: No, it's as if there were trapdoors in the roads, located some 100 m before each intersection. The trapdoor leads to a bottomless hole, and opens automatically whenever there's a queue all the way from the trapdoor to the intersection.

Another important difference compared to the road network is the traffic pattern. The bulk of the traffic on the Internet is transfer of fairly large files. A typical file will be divided into somewhere from a hundred packets for a moderate size image, to several thousands of packets for larger files. All these packets have the same source and destination addresses.

TOM: So when my computer sends a file, it transmits a thousand packets straight away?

NIELS: With the earliest versions of TCP, you might almost have done that. The TCP protocol included flow control from the start; this mechanism lets the receiver tell the sender how much data the receiver can handle, and the sender must not send more than that. In the road traffic analogy, you tell the sender that you have free parking space for only ten trucks, and then the sender will load at most ten trucks and send them your way. When the trucks have arrived, been unloaded, and left again, you tell the sender that you have new parking space available.

TOM: So if the receiver has space for a thousand packets, I send that?

NIELS: That could cause problems for the network. As soon as the capacity of the computers connected to the network outgrow the capacity of the routers, the network suffered "congestion collapse", with drastically reduced performance. This was during the Internet's infancy in the early 1980s[11]. It became clear that flow control was not sufficient. It prevents overloading of the receiver, but not overloading of intermediate routers. Congestion control was needed too.

[11] See RFC 896 for a contemporary source

TOM: So how does that work?

NIELS: The sender maintains a limit, called the *congestion window*, for the number of packets in-flight in the network, i.e., packets that have been sent but not yet acknowledged. Say the congestion window is five packets. Then you will have five packets in-flight somewhere in the network, waiting for an acknowledgement (ACK) saying that a packet has arrived to the destination and exited the network. And you send a new packet only in response to an ACK for some older packet.

Then TCP also has rules for the adjustment of this window size: When the connection is new, the window size is increased by one packet for each received ACK, meaning that you can send two new packets for each received ACK. When a packet is lost, meaning that the network is getting congested, the window size is cut in half, followed by an increase of one packet per roundtrip time. This style of congestion control was introduced in TCP around 1988, and it seems to work fairly well.

TOM: That sounds like really simple rules. Amazing that it works so well. Anyway, there's one other thing I don't understand. You have been talking a lot about IP and IP-addresses. But you never see any IP-addresses, do you? At least all addresses I use are human-readable, e.g., `wikipedia.org`. Are they related in some way?

NIELS: That's the Domain Name System (DNS), which is a huge distributed database. But it's getting late, so if you'd like me to try to explain how that works, let's do that over dinner.

Chapter 1

Congestion control and TCP

WINDOW-BASED congestion control is an Internet work horse. Up to 90% of the traffic is managed by the TCP protocol, used for web browsing, file-sharing, email transmission, and innumerable other applications. The remaining traffic serves applications such as voice over IP, online gaming, and Domain Name System (DNS) service. To a first approximation, Internet traffic can be divided into two types: TCP traffic and real-time traffic.

This chapter describes the basics of TCP congestion control. Interaction with real-time traffic is the subject of later chapters.

1.1 Objectives of congestion control

Let us first review what congestion control is expected to do. It was invented as a response to the experience of congestion collapse in the early Internet. The objectives are:

Avoid network overload: The probability of packet loss due to congestion should be small.

Efficient resource utilization: All bottleneck links should be fully utilized.

Fair sharing: The congestion control mechanism determines how resources are shared between users. The sharing should be predictable and satisfy some reasonable notion of fairness.

React to changes: The network load is constantly varying, there are occasional routing changes, and some (wireless) links have varying capacity and delay. The congestion control must react to these changes and adjust the sending rates. **(FIXME: Better label.)**

Small queueing delays: Both large queues and large queue fluctuations can harm other applications using the network, in particular real-time applications.

The top two items are the ones of critical importance. For the rest of the list of objectives, I think most can agree that these are nice-to-have properties, while the importance and priority can be argued about.

What about current TCP? When used over the ordinary wired network for which it was designed, TCP achieves all but the last one. For low-power wireless links, and for links with large propagation delay or large bandwidth-delay product, TCP also has difficulties achieving efficient resource utilization.

Furthermore, for a congestion control algorithm to be deployable over the Internet, we have some additional constraints:

Scalability: The algorithm must work for huge networks, in terms of the number of nodes, links and data flows. The amount of resources, e.g., memory and computation power, required in any single node in the network, must not grow with the size of the network topology, or with the number of concurrent data flows.

End-to-end principle: As much as possible should be done at the communication end points. In particular, we cannot require network core to keep any per-flow state.

Robustness: Parameters such as number of flows, capacities, and delays vary over ranges of several orders of magnitude, and knowledge of the parameters is very limited. Hence, the algorithm must work despite severe uncertainty about the parameters.

Tunability: The number of tuning parameters should be small, and the effect of each parameter should be predictable and easy to understand. For scalability, the parameters in each node should be tunable based on *local* information. E.g., to tune the parameters of an AQM scheme for a particular link, it should be sufficient to observe local properties such as the queue size and arrival rate at that link.

Incremental deployability: A new algorithm must work in the setting that a subset of end-nodes and routers are upgraded, and flows using the old and the new algorithm share resources. It is not feasible to have a flag day when all end nodes or all routers switch to a new algorithm.

Openness: One of the reasons for the success of the Internet was that the protocols were free to implement for anyone. Most people in the IETF (Internet Engineering Task Force) and W3C (World Wide Web Consortium) communities understand that patent encumbered protocols are not suitable for use in Internet standards. Patenting a network

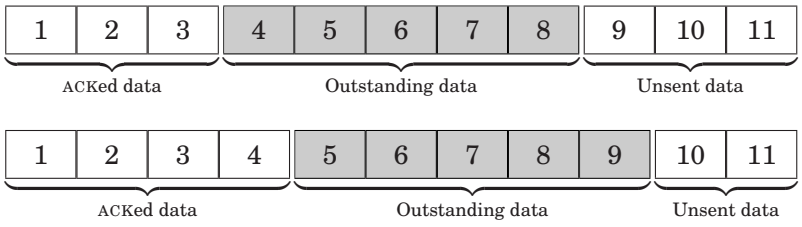

Figure 1.1: Sliding window control. The window size is the amount of outstanding data. In this example, it is five packets. In the top figure, eight packets have been transmitted, and ACKs for the first three packets have been received. The bottom figure illustrates the situation after the ACK for the fourth packet has been received. One more packet, the ninth, is transmitted, and the window of outstanding data "slides" forward one packet.

protocol almost surely guarantees that the protocol will never see any wide use on the Internet[1].

1.2 Additive increase, multiplicative decrease

TCP uses a "sliding window" flow control. The window limits the amount of data that can be sent without waiting for acknowledgement (ACK) from the receiver. When the window is constant, this results in the so called "ACK-clock"; the timing of each sent packet is determined by the reception of the ACK for an earlier packet. This is illustrated in Fig. 1.1.

Window-based congestion control was originally motivated by analogy with physical flows.

> *. . . the packet flow is what a physicist would call 'conservative': A new packet isn't put into the network until an old packet leaves. The physics of flow predicts that systems with this property should be robust in the face of congestion.*
>
> Van Jacobson, 1988

Now, the window size is not a fixed value, it is adapted to the conditions on the network. Now, the sender doesn't know much about other parts of the network, the only information readily available is if (and when) ACKs are received. Omitting some of the details, TCP roughly works as follows: A packet is considered lost if no ACK is received in a reasonable time, or if ACKs for the next few packets are received without any ACK for the packet

[1]For a sad example, see all the great research in secure password authentication. That is an urgent problem which is technically solved, yet nobody is using the solutions because the entire area is a patent swamp.

in question. Lost packets are retransmitted (we will not go into any details on how this is done), but the losses also drives the congestion control.

When a new TCP connection is opened, the initial value for the window size is one or a few packets. These packets can be sent right away. Then, as long as no loss is detected, the window size is increased by one packet for each ACK received. This means that for each ACK received, *two* new packets are sent, and the window size grows exponentially, doubling once per roundtrip time. For historical reasons, this mode of exponential window growth is called *slow start* (it was a slow down compared to the behaviour before TCP had any congestion control at all).

At the first packet loss, the TCP sender switches to *congestion avoidance* mode, using a mechanism known as additive increase, multiplicative decrease (AIMD). The window size is immediately halved, which implies that the sender has to wait until half of the outstanding packets have been ACKed before sending any new data. And then the window size is increased slowly, by one packet per roundtrip time (rather than one packet per ACK as during slow start). This pattern is then repeated: When a packet loss is detected, the window size is cut in half (and the packet in question is also retransmitted), and as long as there are no packet losses, the window size is increased by one packet per roundtrip time.

1.3 Queues

Let us consider the simplest possible case: A network with a single TCP flow, with a single bottleneck on the path. This will shed some light on the basic relationships between window size, queueing, and the AIMD mechanism. We first need to understand a few concepts.

The *roundtrip time* is the delay from the time instant when a packet is sent, until the corresponding ACK gets back to the sender. In general, this will include queuing delay, so let us use *roundtrip propagation time* to mean the roundtrip time *excluding* any time the packet (or the ACK) spends in a queue. The `ping` program measures the roundtrip time. In case the network is ligthly loaded, this will be almost the same as the roundtrip propagation time.

The *bottleneck link* is the link on the path with the smallest capacity. The capacity of the path is the capacity of the bottleneck link. If we gradually increase the sending rate, a queue will start to build up just before the bottleneck. Let us refer to this queue as the *bottleneck queue*.

The pipe size is the capacity of the bottleneck link times the roundtrip propagation time. Let's look at two examples: Over a local network, we may have a capacity of 100 Mbit/s and a roundtrip time of 1 ms. Then the pipe size is about 12 KByte. For a path over a transatlantic cable, the roundtrip time can be 100 ms, and if the bottleneck is an 8 Mbit/s ADSL modem at either end, the pipe size is about 100 KByte. For a high-capacity satellite link, the pipe size can be many times larger than that.

How is this related to TCP and its window size? The ACK-clock keeps the average sending rate of TCP at one window per roundtrip time. If the window size is smaller than the pipe size, then the average rate is smaller than the botttleneck capacity, and network resources are wasted. So we'd like the window size to be larger than the pipe size.

What happens if the window size is larger than the pipe size? Then a queue will build up at the bottleneck. The sending rate over the bottleneck link will be exactly the capacity of that link, and by the ACK-clock, the TCP sender will also transmit packets at the same rate. If we let things settle down, there will be no queues bulding up anywhere else, and that means that the size of the bottleneck queue will be precisely the difference between the window size and the pipe size. So to get small queues, we must aim for a window size slightly larger than the pipe size.

Queues are of limited size; queued packets are stored in a memory buffer, and these buffers are limited in size. When a buffer gets full, arriving packets are not put in the queue, they are discarded, "dropped onto the floor". (In principle, some other packet somewhere in the queue could be dropped in order to make room for the new packet, but that is not the usual way to do it).

Now, with TCP congestion control, as long as no packets are lost, the sender keeps increasing the window size (by the slow start mechanism, or by additive increase). When it reaches the pipe size, the bottleneck queue starts to build up as the window size keeps growing. Eventually, the window size reaches the sum of the pipe size and the buffer size for the bottleneck queue. At this point, the buffer gets full, and some packet is dropped. In response, TCP halves its window size and waits until packets beyond the new window size are ACKed. This waiting time lets the bottleneck link serve queued packets, without any new packets arriving, and the bottleneck queue shrinks.

The TCP window evolves roughly like a sawtooth wave. If we want to maintain full utilization over the bottleneck link over the entire cycle, the queue must never become completely empty. Then the smallest window size during the cycle, i.e., the size just after it has been halved, must exceed the pipe size. This translates into a requirement on the buffer size: The buffer size must be larger than the pipe size (or possibly equal).

Now, for actual Internet routers, the pipe size (and in particuler the end-to-end roundtrip time) is not known in advance. Instead an old Internet rule of thumb says that buffer size in a router should be at least 250 ms times the link capacity, where 250 ms is a reasonable upper bound on end-to-end round trip time. For a router with a capacity of 1 Gbit on each ports, this corresponds to 20 MByte of RAM per port. This rule of thumb has been questioned in recent years. The argument above was based on the case of a single TCP flow; if a large number of flows share a bottleneck link, one might be able to use less buffering and still get full utilization. Anyway, the point I'd like to make here is that buffers are pretty large, and are going to remain large for some time to come.

(FIXME: Summary section?)

To conclude, we see that TCP will fill up the buffers of any bottleneck queue, resulting in queueing delays of up to 250 ms, as well as a (small) packet loss probability. Any non-TCP traffic which share that queue will also experience corresponding delays, delay variations, and occasional packet losses. In the next chapter, we look at how to protect real-time or delay sensitive traffic by differentiating between different types of traffic. Further on, we will look at an alternative congestion control mechanism which can maintain bottleneck queues at moderate size.

Chapter 2

Simple quality of service

The current Internet provides only "best-effort" service: when you send a packet, you don't know how long it will take for it to reach its destination, or even if it will be delivered at all. If some router on the path is overloaded, even briefly, you packet may be discared. In addition, packets which are delivered may be reordered or duplicated by the network.

Small queueing delays are important for the quality of real-time applications. Since the AIMD control in TCP makes bottleneck queues grow until they overflow, the quality of real-time applications is degraded when they share a bottleneck with TCP traffic. For this reason, most backbone links are highly over-provisioned.

In this context the term *Quality of Service* (QoS) refers to any mechanism to extend the Internet architecture to provide something better (and more expensive) than best-effort service for certain applications.

(FIXME: Wording: Consistently use "flow" rather than "connection"?)

2.1 The quality of service dream

In the late 1990s and the early 2000s, there was a lot of interest in end-to-end quality of service, mainly from network operators with telecom background. I imagine they saw a win-win scenario: First, they can charge users extra for premium service. Second, they can postpone expensive link upgrades, and let link utilization approach 100%. If quality degrades for non-premium users, that's an opportunity to get them to pay for premium.

Now, there are a couple of problems. The first problem is that in general, an end-to-end connection involves more than one operator. So if you are an operator and you want to sell a premium service to your customers, you have to negotiate for premium treatment of their packets also by *other* operators. This is may be technically possible, but the additional coordination of the different network infrastructure and policies of competing operators is a problem of gordian proportions.

The second problem is that the circumstances where users are likely to want to pay for higher quality are the same circumstances where high quality is difficult to deliver. And conversely, as long as there's no shortage of network resources, there's no incentive for users to pay for anything beyond best-effort service.

For example, let's consider one group of users who are likely to want the premium service: News photographers. It is important, both for personal recognition and for income, to get photos of breaking news uploaded to an image bureau as soon as possible. So paying for premium service makes good sense. Now, imagine reporters and photographers gathered at some news scene, in the middle of nowhere, where the only network connectivity is via mobile broadband. They all want to send photos and video to their news desk or to some image bureau, and this can be large volumes of data. They have all paid for premium service, but it's simply not possible to make them all happy. Maybe one could auction off available capacity to the highest bidders, but this gets complex and not particularly user friendly.

2.2 Reality

There are several mechanisms in use for differentiating between different traffic classes *within* the network of a single operator or other organization. At the smallest scale, a sole router or ADSL modem can be configured with priorities or rate limits for certain types of traffic.

At a slightly larger scale, a local network can use multiple VLANs to separate between different types of traffic. E.g., one VLAN for IP telephones, and another for office computers.

And within the networks of an organization or network operator, traffic can be classified and differentiated using the Differentiated Services Code Point in the IP header. Or one can use MPLS to explicitly manage allocation of network resources for different traffic types or for different customers.

All these mechanisms have their place, but they don't easily work across organizational boundaries. End-to-end quality of service is still a pipe dream.

2.3 End-to-end quality of service

What do I mean by *end-to-end* quality of service? I mean that the application in the end node tells the network what kind of service it wants, and then the network will do its best to provide this.

The problem is that when the quality of different traffic classes are specified in terms like *best effort* versus *expedited forwarding*, then one class is clearly better than the other. And then why would an application in an end node ever want anything but the traffic class with the highest quality?

One possibility is to charge the end user extra for each packet sent with a higher quality traffic class. This is the quality of service dream, discussed above, with all its problems.

Or one could rely on faith in the end nodes, just like congestion control relies on end nodes to really follow the AIMD rules. But if it's easy to specify the high quality traffic for all packets, and this gives the end user much better service at the expense of other users, then end nodes simply can't be trusted to specify the traffic class. Operators will hardly trust even other operators. This kills end-to-end quality of service, and one is left with an architecture like differentiated services, where the traffic class of a packet is assigned when the packet enters an organization's network, and either reset or ignored when it crosses the next organizational boundary.

To have any chance of deploying end-to-end quality of service, we must define traffic classes in such a way that there's no way for an end node to get a large advantage over other users by bending the rules.

For comparison, let us get return briefly to the subject of congestion control, and note that it relies on faith in the end nodes. **(FIXME: poor security.)** TCP flows are expected to get a fair share of networking resources. However, it's possible for a user to get a larger share by making several TCP connections in parallel, and this is a trick that's actually quite common. In principle, an end node could also get a larger share by replacing the AIMD rules in the local TCP implementation by something different which results in larger window sizes than a conforming TCP implementation.

And the interesting observation here is that such abuses are no big deal. Congestion control over the Internet doesn't break down. A user who tries to do large file transfers completely without congestion control will cause more harm to his own connection than to the Internet at large. Effective denial of service attacks require the cooperation of a large number of end nodes. One fact which most likely helps is that end nodes typically have a too small upstream capacity to do any serious damage.

After this diversion, back to quality of service. Can we define some real-time traffic class which is of use for real-time applications, and which end nodes can be trusted to specify?

2.4 A low-delay traffic class

Let us first review the properties of current "best effort" packet delivery. Routers forward packets. When the incoming rate exceeds capacity, packets are queued for transmission, using a fairy large buffer (remember the rule of thumb, with buffers sized as 250 ms times capacity) to store packets. When the buffer gets full, incoming packets are discarded.

And this is half of the story. The other half is congestion control in the end nodes. Together, the resulting behaviour under load is a small packet loss rate, and potentially large queueing delay.

The idea here is to define a traffic class with complementary properties under load: Low queueing delay, and potentially large packet loss rate. On the router side, this is implemented by using a very small buffer size. To be able handle packets which by chance arrive at different ports at the same time, the buffer at each output port should be able to hold one packet per input port (for routers with a large number of ports, one can perhaps get away with a smaller number, e.g., the square root of the number of ports, since with an increasing number of ports, it gets extremely unlikely to have arrivals on all ports at the same time).

To give some concrete numbers, say the bufffer size is ten packets of size 1500 bytes, and that the link capacity is 1 Gbit/s. Then the maximum queueing delay per router is just 0.12 ms.

The job of the end nodes using this service is to manage their sending rates to keep the packet loss rate under control. Using a congestion control like TCP's will not work well, due to the small buffers. It might be possible to use some different congestion control, but an attractive alternative is to instead use admission control. Let each application have a low and a high threshold for the packet loss rate they can tolerate. At startup, the application measures the loss rate. If it is above the threshold, the application aborts and it has to try again later (any automatic retries should employ exponential back-off). When the connection is established, the application should monitor the loss rate, and disconnect if it exceeds the high threshold. The choice depends on the application, but reasonable values would be one or a few percent for the low threahold, and up to 10% for the high threshold.

With this combination of small buffers and admission control, we get a service with the following properties under load: Low delay, moderate packet loss, and a blocking probability.

Which applications would want to use this service? First note that for any application which needs to resend lost packets, this low-delay traffic class is not of much use. The loss rate under load will be higher than for the low-loss traffic class, and the delays from additional retransmissions will outweight the gains of less queing delays for the delivered packets.

Voice over IP can take advantage of the low-delay class, provided that one uses some additional redundancy or error correction coding to compensate for lost packets. The application can adapt to different packet loss rates, but the the sending rate must not be increased in response to higher packet loss. Instead, increased channel coding (more redundancy) should be compensated for by also adapting the source coding to generate fewer data bits (lower sound quality).

Similarly, online action games can take advantage of the low-delay class, if the protocols are designed to tolerate some lost packets. E.g., it's better to have each packet encode the positions of recently moved objects, than to just enode the change of position as each object moves.

One obvious application is clock synchronizaton protocols such as Network Time Protocol (NTP) and the Precise Time Protocol (IEEE 1588, also

know as PTP). These protocols can easily handle high packet loss rate, while packet delay shows up as measurement errors, which are difficult to filter out when the network is loaded.

2.5 Priority between classes

Finally, we have to describe the relative priority of the low-loss and the low-delay traffic classes. Routers will place the packets into separate queues, with different buffer size. Each time the outgoing link is ready to transmit a new packet, the router must decide which queue should be served. It is tempting to say that the low-delay packets must never wait for a low-loss packet, and hence the low-delay queue should be given the highest priority. But then there is a potential problem: We could get in the situation where the arrival rate of the low-delay is almost the same as link capacity, and then the low-loss traffic would become starved for capacity. To avoid this situation, the router could impose a rate limit on the low-delay traffic (which should be active only when the queue for low-loss traffic is non-empty).

A simpler alternative would be to use round-robin scheduling between the two queues, possibly also taking paket size into account. **(FIXME: Do we need to get into any details on the scheduling? This ought to be a solved problem in the literature.)**

With admission control based on probing, there's also a risk that probe packets will disturb the admitted flows. The problematic case is a loaded network where the packet loss rate of the low-delay class is sufficiently high that most new connections send a stream of probe-packets, and then abort. Then if there are a lot of new flows, the amount probe packets will cause additional dropped packets also for the admitted flows.

For this reason, it makes sense to introduce yet another traffic class for probe traffic, which is given lower priority than the low-delay class, but is otherwise treated identically and, at least in principle, put in the same queue. If the buffer is full when a non-probe packets arrive, then if possible the router should discard enqueued probe packets to make room, rather than discarding the newly arrived packets. **(FIXME: Or are things like this usually implemented using different queues and some extra book-keeping? It might be impractical to discard packets in the middle of a queue?)**

An application starting a new flow can then either send a stream of probe packets to determine if there is room in the network. Or it can be optimistic and start sending real data righht away, but using the probe traffic class. After measuring the resulting loss rate, it can upgrade the traffic class to the low-delay class if the loss rate is small enough, or abort if the loss rate is too high.

Further implementation issues are deferred to Chapter 5.

Chapter 3

Window-based congestion control

WINDOW-BASED congestion control can be seen as cascaded control system with two feedback loops. The inner-loop determines when each packet is transmitted, and hence the sending rate. The feedback signal is the arrival of ACKs, and the window size is a controller input. The outer-loop adjusts the window size, based on the received ACKs and any other available feedback information from the network.

(FIXME: Add block diagram.)

(FIXME: Better title? This chapter focuses on the inner-loop.)

3.1 Stability

The first question to ask about the inner loop is whether or not it is stable. Assume that each flow across the network has its window size set to some constant value. What can we say about the time evolution of the queue sizes, assuming they start at some arbitrary initial values? It is clear that no queue size can diverge to infinity, since the total number of packets is constant, namely the sum of all the window sizes. For each queue, the queue size is limited by the sum of the window sizes for those flows which traverse the queue in question. So in one sense, the system is obviously stable.

Now, this doesn't necessarily mean that the system is *asymptotically stable*, i.e., that the size of each queue will converges to an equilibrium value. There may be some oscillations where two (or more) queues alternate in size. I'm not aware of any rigorous studies of this possibility in a fully general network; but to me having any sustained oscillations seems unlikely. In my thesis, I was able to prove stability (using the queueing model described in Section 3.3) in the following special cases:

- The case of a single bottleneck and a single flow with an arbitrary propagation delay, is globally asymptotically stable.

- The case of a single bottleneck and an arbitrary number of flows with arbitrary delays, is locally asymptotically stable.
- With an arbitrary network topology, under the approximation of no signalling delays, the system is globally asymptotically stable.

(FIXME: Full citation?)

These results indicate that by themselves, neither delay, aggregation, or complex network topology, leads to any stability problems. To me, sustained large oscillations of the queue sizes seem unlikely.

3.2 Inner-loop behaviour

In queueing theory, the behaviour of a queue is derived from its arrival *rate*. Essentially, whenever the arrival rate exceeds capacity, the queue grows accordingly, and when the arrival rate is below capacity, the queue shrinks (unless it's already empty). Therefore, to describe the queue behaviour under window-based congestion control the first problem is to describe how window size and sending rate are related. As we have seem in Chapter 1, the average sending rate is one window per roundtrip time, but to describe the *dynamical* behaviour, we need something a bit more precise.

We return to our basic example, with a single flow using window-based congestion control, and a single bottleneck link. It turns out that the behaviour can be quite different, depending on what other traffic is using the bottleneck link. So let us also assume that some fraction of the capacity of the bottleneck is used by cross traffic which is of constant rate, with no congestion control (e.g., some number of voice over IP calls, or some other real-time data streams). Also assume that our window size is large enough to get make the queue non-empty, and get full utilization of the bottleneck.

Now, assume that we start with a network in balance. Our flow has a sending rate of one window per roundtrip time, and the queue delay is precisely large enough to make the sum of our flow's rate and the cross traffic arrival rate equal capacity. Now, what happens if our sender increases the window size by one packet? One additional packet is transmitted, which arrives to the bottleneck and makes the queue grow by one packet.

If there is no cross-traffic, then that's all that happens: there's no change in the rate our packets are sent out on the link (they're sent at link capacity), hence no change in the rate of received ACKs. In this case, the queue size is essentially the window size minus the pipe size, with no dynamics beyond the time delay for transmitted packets to arrive at the bottleneck. **(FIXME: Relation between TCP "cwnd" and the number of outstanding packets, which we call the "window size")**.

On the other hand, if there is cross traffic, then the additional packet gets in front of cross traffic packets. We get an additional packet transmitted on the link, and the next packet in our flow is delayed only by the

transmission time, which is smaller then the inter-packet time of the flow. As a result, we get back ACKs at a higher rate, and will then use a slightly higher rate than we started with for several roundtrip times. During this time, the total arrival rate to the bottleneck is above capacity, the queue grows, and the cross traffic will be served at a rate slightly below its arrival rate (instead having its packets wait longer in the queue). The queue grows until the rate of one window (of the new size) per roundtrip time (with a new longer queueing delay) is the same as the rate we started with.

We may need to introduce a few equations to make this clear. Let τ denote the roundtrip propagation time (i.e., excluding queueing), let c denote the capacity of the bottleneck link, and let γ denote the proportion of the capacity which is available for our flow, i.e., not consumed by cross traffic. Let w denote the window size and q denote the size of the bottleneck queue. Also assume that $w > \gamma c\tau$. The total roundtrip time is then $\tau + q/c$.

Then we have balance, or a *system equilibrium*, when $q = w/\gamma - c\tau$. So when $\gamma = 1$ (no cross traffic), the queue size is the difference between window size and pipe size, as we have seen earlier. Then, if the window is increased by one packet, the queue size also increases with one packet. On the other hand, if $\gamma < 1$, then if the window is increased by one packet, the queue size increases by $1/\gamma$ packets.

3.3 Inner-loop dynamics

Appendix A describes a model for the dynamic relationship between window size and queue size. As in the previous section, τ is the roundtrip propagation time, c is the bottleneck capacity, w is the window size and q is the queue size. The rate of change of the latter two variables are denoted $\dot{w}$ and $\dot{q}$. For simplicity, we assume that the forward delay (i.e., the propagation delay between the sender and the bottleneck) is zero. Like in the previous section, we also assume that we have cross-traffic of constant rate, leaving a proportion γ of the capacity available for our flow. In this setting, the model is

$$\dot{q}(t) = \frac{w(t-\tau)}{\tau + q(t-\tau)/c} + \dot{w}(t) - \gamma c \tag{3.1}$$

(FIXME: Mention non-negativity constraint?) The model describes a system where the window size is the input signal, and the queue size is the output signal. **(FIXME: Repeat stability.)**

(FIXME: Include some step response figures, from thesis/simulations.)

In automatic control, after stability, two of the most important system properties are the static gain and the convergence time. If the system is in equlibrium, and we make a small change to the window size (the input signal), how long does it take until the queue size (the output signal) settles at a new value? That delay is the system's convergence time. And how

large is the change of the output in relation to the change of the input? That is the static gain.

We have already seen that the static gain is $1/\gamma$. Bounds for the system time constant are derived in my thesis, and can be summarized as follows. **(FIXME: Citation?)**. When $\gamma > 0.3$, i.e., when the cross traffic consumes at most 70% of the bottleneck capacity, the time constant is at most 3.4 times the roundtrip time, which corresponds to a convergence time of roughly 10 times the roundtrip time. Remember that the roundtrip time includes queueing delay, which may be large if the window size is large.

On the other hand, if $\gamma \leq 0.3$, i.e., cross traffic consuming the lion's share of the capacity, then the time constant is approximately RTT/γ. **(FIXME: If we have to use the RTT abbrev, introduce it somewhere.)**

3.4 Lessons for outer-loop design

Window based congestion control is an outer loop which adjusts the window size depending on whatever information can be inferred about the state of the network. So this is an example of cascade control, where the window size is a reference input for the inner loop, and the control signal for the outer loop. **(FIXME: Figure? Rewrite paragraph?)**

First, I'd like to reiterate that the inner loop is stable. So it's *not* the job of the outer loop to stabilize the system. It just needs to take inner loop dynamics into account and be careful not to destabilize it.

Second, the dynamics depends on the amount of cross traffic. If we compare a 10 Mbit/s bottleneck with no cross traffic to a 100 Mbit/s bottleneck with 90% cross traffic, the available capacity (denoted γc in the previous section) is the same, 10 Mbit/s. The static gain, from change in window size to change in queue size, differ by a factor of 10, but this may be of less importance since this difference vanishes if we scale queue size by link capacity: The time for transmitting one packet over 10 Mbit/s is the same as the time needed to send ten packets over 100 Mbit/s.

A more crucial difference is the time constant, with no cross traffic, the queue converges to the new size in less than one roundtrip time, while with 90% cross traffic, the time constant is 10 roundtrip times, resulting in a convergence time a few times larger than that.

Third, a rule of thumb for cascade control is that the outer loop should operate on a time scale ten times slower than the inner loop. With this separation of time scales, the outer loop design can be based on the static gain of the inner loop, ignoring the dynamics. But it seems a bit difficult to arrange for this separation of time scales, and at the same time support high levels of cross traffic, since the time constant of the inner loop approaches infinity as γ approaches zero.

To get a well behaved system for arbitrary delays and arbitrary levels of cross traffic, it seems necessary to take the inner loop dynamics into

account.

Chapter 4

A new congestion control

USING multiplicative decrease, halving the window size, makes sense in response to packets being lost due to congestion: When the network is being overloaded, dropping packets, it's too late for more fien grained adjustments.

But what if we can get some reliable information on queue size earlier? This chapter explores an alternative congestion control mechanism. aim of the new protocol is to maintain the efficiency and fairness properties of TCP, but with significantly smaller bottleneck queues.

4.1 Structure

Like TCP, the new protocol is window-based. The ACK-clock is a simple and efficient per-packet rule which controls the sending rate. We have seen in the previous chapter that inner-loop is stable, for arbitrary propagation delay in the network. We just modify the control of the window size. **(FIXME: Figure)**

Similarly to ECN, we let routers on the path set a bit in the packet header depending on the queue size. The receiver copies this bit into the corresponding ACK, which gives the sender information about the state of the network. Standard ECN requires the sender to respond to such mark bits in the same way as to a lost packet, i.e., by halving the window size. We depart from this, and let senders react more softly to the mark bits, and we can then allow mark bits to be set more frequently.

The proposed scheme is structurally very similar to standard TCP combined with explicit congestion notification (ECN) and active queue management (AQM), with some subtle but crucial differences in the handling at both end hosts and routers. **(FIXME: Figure?)**

Additive increase

The lack of feedback in the absence of congestion, and the desire to have efficient utilization of resources, requires some mechanism that increases the sending rates in the absence of congestion. This growth rate is going to be one of the system parameters, and for stability reasons, it makes sense to scale it down with the roundtrip time. Hence, we keep the additive increase part of TCP in our new protocol: in the absence of congestion, increase the window size by oen packet per roundtrip time.

With the flow analogy in mind, additive increase can be thought of as a pressure applied by the sources, which forces more fluid into the system, filling up the pipes, and when the pipes are full, the pressure causes queues to grow. For an effective congestion control, we must design a way for the queues to impose a back-pressure on the fluid.

Additive decrease

With window control according of TCP, feedback is a relatively rare event, and the response to each feedback event is strong. The TCP "square root formula" says that

$$\text{rate} = \frac{\text{packet size}}{\text{roundtrip time}} \sqrt{\frac{2(1-p)}{p}} \tag{4.1}$$

This formula is a consequence of the AIMD window adjustments: it doesn't matter if p is the packet loss probability, or the probability that routers set the ECN mark bit.

Since the rate can also be expressed as one window per roundtrip time, we can turn the formula around to express the probability p as a function of the window size. If we let the window size be n packets, and in addition we assume that p is small, we get $p \approx 2/n^2$. The average interval between feedback events is then $n^2/2$ packets or $n/2$ roundtrip times. Hence, the sender will operate for many RTTs between feedback events, growing its window according to the additive increase law.

We aim for more frequent signalling, on the average of one feedback event per RTT. To get there, we need a softer response to each event: For each received ACK carrying a congestion indication, the window size is decreased by one packet. With this rule, senders will no longer operate in open-loop for long time periods.

Another important benefit is that from a router's point of view, the system becomes more predictable. Under the one-packet decrease rule, when a router decides to set a mark bit, the effect is that one packet less will be arriving a round-trip time later. For comparison, if we use ECN with multiplicative decrease, the corresponding flow will halve its window. Since the window size is unknown, in this case both the timing and the *magnitude* of the effect is unknown. With the one-packet decrease rule, the magnitude of the response is known, only the round-trip time is uncertain.

One can also reason about the amount of information the sender gets from the mark bits. For the AIMD rule, the amount of information one gets about the network state per roundtrip time is inversely proportional to the window size. In contrast, with the one packet decrease rule, the amount of information per roundtrip time is independent of the window size. See Appendix B for the details.

Congestion feedback

The amount of signalling that can be sent from the network to the sources is limited. One common approach is to piggyback a congestion indication on packets forwarded to the receiver, which copies the information into the corresponding acknowledgment and sends it back. One subtle advantage with this approach is that if the congestion signal is lost, that implies that a data packet (or acknowledgment) is lost too, which is going to be noticed by the sender. This is unlike the source quench messages that were used in the early days of TCP, and transmitted as independent Internet Control Message Protocol packets.

We will assume that we can allocate one bit of information in the packet header, analogous to the standardized ECN bits. The main difference to ECN is that the response we are defining for the sender is different, and also the rules for setting the bit are different.

One useful consequence of doing the feedback as a per-packet marking is to get a signal that is proportional to flow size. We can then get a fair sharing between the flows, without routers needing to be aware of which packets belongs to which flow.

Only bottleneck routers should add marks to forwarded packets, for several reasons

- The state of non-bottlenecks is generally not interesting enough to spend signalling resources on.
- There are usually a small number of bottlenecks along a given path. Combining the signalling from several routers is somewhat difficult both for implementation and for analysis, so the smaller the number, the better.
- For deployment, it is desirable that the new congestion control can be deployed by upgrading only bottleneck routers, and leave the large number of non-bottleneck routers unchanged.

One consequence is that a network where no link is fully utilized will generate no packet marks. We thus get no quantitative information about how close the network is to full utilization.

Marking rule

It is important that the feedback rule uses a minimal number of parameters; one of the problems with current AQM is that the algorithms are difficult to tune so that they work well. We are going to consider rules with a single parameter.

For good control performance, delay is important. If the state is estimated using an explicit observer (at either sender or router), time constants must be kept small. For this reason, there is limited room for sophisticated estimation in the router. We generate feedback directly from the instantaneous queue size, which is the primary value we want to control.

We will describe the feedback process as a stochastic procedure, where each packet is marked with some probability which depends on the current queue size. The marking must not necessarily be implemented in a stochastic way, e.g., it may be possible to ensure that the right proportion of packets are marked, using virtual queues and some fully deterministic rules. In the latter case, the marking process will still appear to be stochastic when observed by sources, given that the cross-traffic is random.

The feedback to be attached to a packet can, in principle, be attached at any time the packet is in the queue. In practice, the most common alternatives are to mark packets either at arrival to the queue, or just before they are transmitted. Let us add the marks to packets at exit from the queue, since this choice minimizes the signalling delay.

The precise relation between queue size and mark probability is somewhat arbitrary. We want a function which is zero when the queue is empty, increases with the queue size, tends to one when the queue gets very large, and which is simple to describe. It also seems natural to use a concave function (i.e., decreasing slope). Let us choose the simple one-parameter rule

$$p(q) = \frac{q}{q_0 + q}$$

where the q_0 parameter corresponds to the size of a severely congested queue, where 50% of the packets are marked. Hence q_0 can be much larger than the desired queue size. A large q_0 implies a small gain from q to p. **(FIXME: Needs figure!)**

4.2 Protocol summary

Let us summarize the protocol actions. The sender maintains its window size, the number of outstanding packets, and transmits new packet in response to ACKs (except when the window is adjusted, see below). Packets carry a mark bit. Each router on the path sets that bit with a probability depending on the router's current queue size. The reciever copies the mark bit from each received packet into the corresponding ACK packet.

When the sender processes each ACK, it also updates the window size. The window is increased by one packet per each round trip time, and it is decreased by one packet per received ACK which has the mark bit set (except that if the window size is one packet, it is not decreased).

Actual packet losses indicate sever congestion, or congestion at bottleneck not supporting this packet marking. Losses should be detected and responded to in the same way as in standard TCP.

We also need some rule to switch from the slow start mode to congestion avoidance mode based on received marks bits, in the absense of packet losses. This has not been worked out, but one way is discussed in Chapter 5.

4.3 Analysis

Consider the single source, single bottleneck case. Assume constant cross traffic, using a proportion γ of the bottleneck capacity c, and for simplicity let the forward propagation delay be zero. Parameters are the packet size m, used for the additive increase, and q_0, used for the marking probability. This system can be described by the following equations:

$$\dot{q}(t) = \frac{w(t-\tau)}{\tau + q(t-\tau)/c} + \dot{w}(t) - \gamma c$$
$$\dot{w}(t) = \frac{1}{\tau + q(t-\tau)/c}\left(m - \frac{q(t-\tau)}{q_0 + q(t-\tau)}\, w(t-\tau)\right)$$

The first equation is the queue dynamics, from the model in Appendix A. The second equation is the window update law, where the first term is the additive increase, one packet per roundtrip time, and the second term is the additive decrease, which is the product of the ACK-rate and the marking probability.

In Appendix D, it is shown that in the single-flow single-bottleneck topology, this system is stable, provided that $q_0 \geq 4c\tau$.

(FIXME: What else here? More on the equilibrium values? Fluid flow simulations?)

4.4 Evaluation

There's a very large number of proposed variants of TCP. Most of the recent ones are not intended to reduce queueing delay, instead, the aim of most current TCP research is better preformance over networks path with a very large pipe size, or better preformance over wireless links (characterized by varying bandwidth and delay, and packet losses which are not caused by congestion). For this reason, we compare to two older TCP versions: TCP New Reno, and TCP Vegas.

TCP New Reno is the most widely used version of TCP, officially elevated from Experimental status to a Proposed Standard in 2004. **(FIXME:**

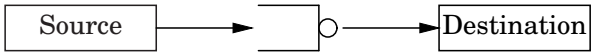

Figure 4.1: The simplest possible topology. A single data flow, with a single intermediate bottleneck router.

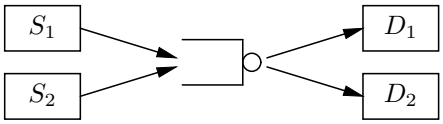

Figure 4.2: Topology with a single bottleneck shared by two flows.

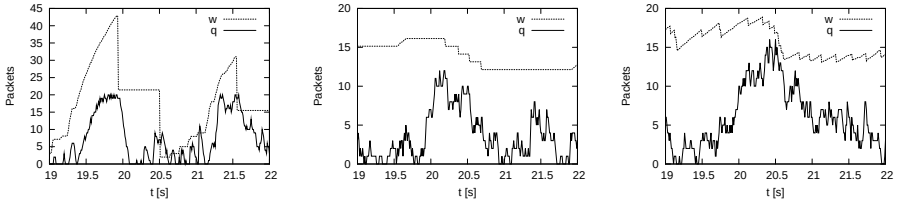

Figure 4.3: Window size (w) and bottleneck queue size (q) for a short segment of a simulation with a single bottleneck and a congestion controlled flow sharing the bottleneck with Poisson cross-traffic. At $t = 20$ s, the cross-traffic intensity is increased from 20% to 40% of link capacity. Left to right: New Reno, Vegas, and the new protocol. Note the different scale on the vertical axis.

RFC 3782.) TCP Vegas uses delay-based congestion control, using measurements of the roundtrip time to react when queues in the network grow. It is relevant because one of the design goals was to reduce packet losses, by reacting to congestion before queues overflow. More recent delay-based congestion control methods, in particular Fast TCP, are aimed for connections with large pipe size, which is a different problem. **(FIXME: How much motivation and background is really needed here?)**

(FIXME: Material below copied from a draft journal paper. Needs work.)

We compare the results to TCP New Reno and TCP Vegas. We are interested both in end-to-end properties, such as throughput, packet loss rate, and delay, and per-link properties such as queue size, link utilization and link loss rate.

Single link, single flow

The first scenario uses a single TCP flow over a single bottleneck. The topology is illustrated in Fig. 4.1. The bottleneck capacity is 2 Mbit/s, and the roundtrip propagation delay (excluding queueing delay) is 100 ms. This implies a bandwidth delay product of 16 packets. The buffer size is set to

	New Reno	Vegas	New protocol
Link loss rate (%)	4.27	0.00	0.02
TCP loss rate (%)	5.03	0.00	0.01
Utilization (Mbit/s)	1.64	1.88	1.90
TCP throughput (Mbit/s)	1.02	1.39	1.40
Queue average (packets)	7.26	2.90	3.77
std. dev. (packets)	7.12	2.46	3.41
Forward delay average (ms)	114.66	75.78	80.84
std. dev. (ms)	40.11	13.74	19.61

Table 4.1: Results for the single link, single flow scenario.

	New Reno	Vegas	New protocol
Loss rate (%)	1.56	0.00	0.00
Utilization (Mbit/s)	2.00	2.00	1.99
Queue average (packets)	19.27	5.79	4.58
std. dev. (packets)	6.00	1.91	2.32

Table 4.2: Results for a single link and two flows, per-link values.

	New Reno		Vegas		New protocol	
Flow	#0	#1	#0	#1	#0	#1
TCP throughput (Mbit/s)	1.22	0.38	0.82	0.78	1.17	0.42
TCP loss rate (%)	1.06	3.31	0.00	0.00	0.00	0.00
TCP window (packets)	14.96	7.77	4.28	9.37	5.81	5.21
std. dev.	4.37	2.95	0.47	1.49	1.16	1.20
Forward delay average (ms)	135.55	178.34	50.79	92.95	45.58	88.53
std. dev. (ms)	37.33	35.61	11.15	11.09	13.76	13.41

Table 4.3: Results for a single link and two flows, per-flow values.

21 packets, and q_0 is set to 32 packets. The simulation runs for 60 s, with a file transfer starting at time 0.1 s. The bottleneck link is shared with Poisson cross-traffic, 20% of the capacity, increased to 40% during the interval 20–40 s. The response to this step is illustrated in Fig. 4.3.

Table 4.1 shows average properties. Compared to TCP New Reno, the new mechanism almost eliminates packet losses, while at the same time the TCP throughput is increased and the queueing delay is reduced. The performance is similar to the performance with TCP Vegas. We can also note that TCP New Reno suffers a packet loss rate slightly higher than the link average. The packet loss rate seen by the cross traffic flow is lower, 2.6%.

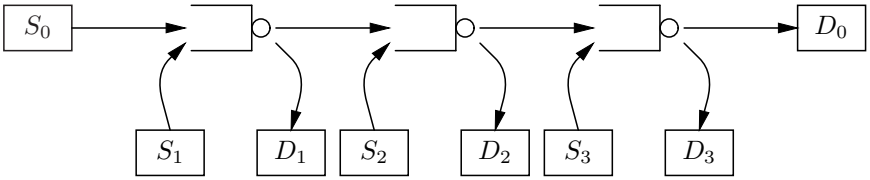

Figure 4.4: The "parking-lot" topology, with three bottlenecks in series.

	New Reno			Vegas			New protocol		
Loss	1.96	1.99	1.67	0.25	0.14	0.09	0.00	0.00	0.00
Util.	2.00	2.00	2.00	1.99	1.99	1.99	1.99	1.99	1.99
Queue	11.98	12.07	11.94	8.78	8.45	8.58	4.37	4.43	4.33
dev.	4.32	4.37	4.35	4.02	3.92	3.71	2.15	2.16	2.09

Table 4.4: Results for the "parking-lot" topology, per-link values.

Single bottleneck, two flows

For the next scenario, we keep the same 2 Mbit/s bottleneck link with 20% Poisson cross-traffic, and start two flows in parallel. The topology is illustrated in Fig. 4.2. The first flow has a roundtrip propagation time (excluding queueing delay) of 20 ms, and the second flow has a five times larger delay, 100 ms. The buffer size is set to 30 packets, and q_0 is set to 15 packets.

Average per-link quantities are summarized in Tab 4.2. Here, the new protocol and TCP Vegas still look similar, reducing loss and delay as compared to New Reno. Average per-flow quantities are summarized in Tab. 4.3. The sharing of capacity is roughly the same for the new protocol and New Reno; the flow with the longer delay gets approximately a third of the throughput of the flow with the shorter delay. With Vegas, the two flows get almost the same throughput. If the packet loss rate were the same for all flows, then both for the new protocol and for New Reno, the two flows would have the same equilibrium window size. However, this is not quite true for New Reno. We see that with New Reno, the long flow sees a three times higher packet loss rate than the short flow, and the long flow uses a smaller average window size.

"Parking-lot" topology

In the final scenario, we have three bottleneck links in series, illustrated in Fig. 4.4. The links are still 2 Mbit/s, with both the buffer size and q_0 set to 20 packets. Over each link, there is one TCP flow, and 20% Poisson cross-traffic. These short flows have 20 ms RTT excluding queueing. We also add one long RTT flow, traversing all three bottlenecks, with an RTT of 150 ms excluding queueing.

	New Reno		Vegas		New protocol	
Flow	#0	#1	#0	#1	#0	#1
Throughput	0.11	1.49	0.88	0.71	0.12	1.48
Loss rate	6.55	1.69	0.37	0.31	0.00	0.00
Window	4.49	13.05	25.48	4.40	2.94	7.06
dev.	2.01	3.57	10.08	0.75	0.75	1.26
Delay	323.05	91.29	270.83	62.56	181.30	44.63
dev.	46.30	26.29	51.95	24.14	23.59	12.49

Table 4.5: Results for the "parking-lot" topology, per-flow values.

Average per-link quantities are summarized in Tab. 4.4. We see that Vegas reduces queueing delay a bit compared to New Reno. While it reduces packet losses, there is still a significant packet loss (different for the three links). The new mechanism reduces queueing further, and it eliminates packet losses. Average per-flow quantities are summarized in Tab 4.5. Flow #0 is the long flow, and flow #1 is one of the short flows. We see that the end-to-end delay and delay jitter is reduced considerably. If we compare the throughput between the long flow and the short flows, we see that the sharing is similar for the new protocol and New Reno; a short flow gets ≈ 13 times the throughput of the long flow. For Vegas, the picture is very different. Vegas aims to give all flows the same throughput, and in this example, the long flow gets slightly higher throughput than a short flow.

(FIXME: Discussion on why the new protocol wins in this topology.)

Chapter 5

Implementation issues

THIS chapter is essentially a draft specification, intended to be detailed enough for use by anyone who wants to do experiments with the mechanisms described in this book.

Since we allocate some values which are reserved for experimental use, implementations based on this chapter must be used on isolated lab networks only. There are a few additional things which must be addressed for experiments (as well as real deployment) over the public Internet. First, all type values and the like must be properly allocated with the Internet Assigned Numbers Authority, and they will then get numerical values different from the ones listed here. Then, it would be prudent to include some mechanism analogous to the ECN nonce (RFC 3540), to detect broken or misbehaving handling of the congestion marks. Finally, there may be some tricks to reduce the overhead.

Up until now, we have talked about the window size as the number of outstanding packets. But in the TCP protocol, the window and related quantities are measured in bytes, and packets may be of varying sizes. Furthermore, we don't count the size of the complete packet, only of the TCP payload, referred to as a TCP segment. So from now on, windows are in bytes, not packets.

5.1 IP header bits

Both mechanisms proposed in this book, the end-to-end traffic classes of Chapter 2, and the new congestion control in Chapter 4, needs to use a few bits in the IP packet header.

Ideally, bits in the old Type of Service field could be reallocated for this, but currently these bits are used for related but incompatible purposes: The explicit congestion notification bits, which specifies that senders respond in the same way to both marked ACKs and real packet losses. And the Differentiated Services Code Point, which is specified to make sense

Value	Name	
0x1e	E2E_QOS	Hop-by-hop option for traffic class
0x32	SOFT_ECN	Hop-by-hop option containing mark bit
0x01	TC_LOW_DELAY	Traffic class for low delay traffic
0x02	TC_LOW_DELAY_PROBE	Traffic class for probe packets
0xfd	SOFT_ECN_SUM	TCP option for congestion signalling

Table 5.1: Magic protocol values.

only within a single organization or *diffserv domain*, and modified as packers cross organizational boundaries.

So the next best way is to use an IPv6 hop-by-hop option. It's questionable if it's worth the effort to support the new mechanisms over IPv4, so let us restrict attention to IPv6. There are type codes reserved for experiments **(FIXME: RFC 4727. Citation?)**. We use codes with the top two bits zero, which means routers not recognizing the option should ignore it.

For experiments with end-to-end traffic classes, we define the type E2E_QOS = 0x1e (third most significant bit zero says that the value is does not change en-route). We then allocate a one octet data field for the end-to-end traffic class, where we define 1 to mean low-delay traffic, and 2 to mean low-delay probe traffic.

For experiments with the new congestion control, we define the type SOFT_ECN = 0x3e (third most significant bit one means that the value can change en-route, and should be replaced by zero when computing a hash of the packet for authentication purposes). We then allocate a one octet data field where the least significant bit is the mark bit. The presence of this option indicates to routers that the flow supports the mechanism and will respond properly to set mark bits.

Both options has a data field of one octet, and no alignment requirements. If both are used (which is unlikely to be useful), the E2E_QOS option should be placed first.

The magic values are listed in Table 5.1.

5.2 TCP header

For the soft ECN mechanism, the receiver must communicate received mark bits back to the sender. We use a TCP option for this purpose, and define the type SOFT_ECN_SUM = 0xfd which is available for experiments. We could let the receiver just copy the mark bits from the IP header of received TCP segments into the corresponding ACK. However, the sender may not receive an ACK for each transmitted packet. Receivers may choose to send fewer ACKs, and even if the receiver sends an ACK for each received segment, some ACKs may get lost. **(FIXME: Say something about how standard ECN does this, with the cwr bit to acknowledge reception of the ecn echo bit.)**

To address this problem, and also deal with packets of different sizes, we use a field containing the sum of the segment sizes of all received packets with the mark bit set. Since the field is of limited size, it can course wrap around, but if it's large enough, the sender can detect that without ambiguity. For now, we use a 32-bit field to avoid problems (although it would be desirable to compress this down to 16 bits).

5.3 Router operation

A router supporting both mechanisms needs three queues at each outgoing link: One queue for real-time traffic, one queue for traffic using soft ECN, and one queue for other traffic.

If an incoming packet has the E2E_QOS hop-by-hop option with a value of TC_LOW_DELAY or TC_LOW_DELAY_PROBE, it is put into the real-time queue (in the unlikely case that the packet also carries a SOFT_ECN option, it is still put into the real-time queue). This queue has *small* maximum size, one packet per incoming link. If a non-probe packet arrives when the queue is full, and there are probe packets in the queue, probe packets are discarded as needed to make room. Otherwise, arriving real-time traffic are simply dropped if the real-time queue is full.

If an incoming packet has the SOFT_ECN option, it is put into the soft ECN queue. The mark bit in this option is set with a probability determined by the current size of this queue, as described in Section 4.1: $p(q) = q/(q_0 + q)$. **(FIXME: Work out integer only implementation. Sketch:** $s \leftarrow s+q$**, if** $s > q_0+q$**, set mark bit and set** $s \leftarrow s-(q_0+q)$**.)** The parameter q_0 should be on the same order of magnitude as the product of the link capacity and expected round trip delay of flows using the link, while the maximum size may be smaller **(FIXME: Refer to tuning advice.) (FIXME: Say something more about maximum queue size. Depends on how the slow-start is done.)**

Packets which carries none of these hop-by-hop options are put in the queue for other traffic, which should behave more or less like queues in current routers: Large size. Usually drop tail, but it could use some variant of active queue management and ECN, if desired.

The queues can be served in a simple round-robin fashion. This should be sufficient to ensure that no type of traffic is starved. More sophisticated methods to control the proportion of capacity given to each traffic type may also be used.

5.4 Real-time applications

A real-time application with a very low sending rate, e.g., an NTP daemon, can just set the E2E_QOS hop-by-hop option on outgoing packets, with the value TC_LOW_DELAY, and hope for the best.

A real-time application with a larger sending rate, e.g., a voice over IP applications, must use acknowledgements so that it can monitor the loss rate over the network, and keep a configured low and high threshold. When setting up a new flow, it should start by sending packets of type TC_LOW_DELAY_PROBE at the same rate as it intends to use during the flow. These packets may or may not contain useful application data. After some 20–100 packets have been sent, the loss rate is estimated. If it above the low threshold, it must abort. If the loss rate is tolerable, it can switch the traffic type to TC_LOW_DELAY. It should keep monitoring the loss rate, and abort if it gets above the high threshold.

As the loss rate varies, applications are encouraged to adjust their coding parameters, but they must not increase the sending rate without probing. If an application wants to increase its rate, it must start by sending the additional packets as probe packets, and switch to non-probe packets only if the loss rate for the probe is below the low threshold.

5.5 TCP sender operation

A TCP sender should set the SOFT_ECN option in the IP header the initial data packets. If the corresponding ACKs don't carry a SOFT_ECN_SUM TCP option, it must not set SOFT_ECN on subsequent packets, and fall back to the traditional AIMD congestion control. The rest of this section applies only if the ACKs carry the SOFT_ECN_SUM TCP option.

The sender keeps a local "soft ECN" count, initialized to zero. When an ACK for new data arrives, the value in the SOFT_ECN_SUM option is compared to the counter, and the difference is computed (taking wraparound into account). If the values differ, the window size is reduced, and the received value is copied to the local count. The sender also applies the usual additive increase, incrementing the window size by one packet per roundtrip time. **(FIXME: Explain that this is "congestion avoidance", and spell out the rules in a little more detail)**.

For the flow startup, as long as the value in the received SOFT_ECN_SUM is zero, the sender can operate in traditional slow start mode: Increase the window size by one packet per ACK, i.e., responding by two new packets to each ACK, in effect doubling the window size each roundtrip time. As soon as the first soft ECN indicatino is received (SOFT_ECN_SUM non-zero), slow start ends and the sender enters congestion avoidance mode.

At the same time, packet losses are detected using the same timeouts and duplicate ACKs rules, with a multiplicative decrease of the window size when a loss is detected.

5.6 TCP receiver operation

Receiver operation is simple. If received packets carry the SOFT_ECN option in the IP header, the receiver keeps a soft ECN counter, initialized to

zero. For each packet where the mark bit in this option is set, the size of the TCP segment in that packet is added to the counter. Each ACK sent should carry the SOFT_ECN_SUM option in the TCP header, with the value copied fromt he current counter. **(FIXME: Clearer naming of the send and receive counters.)**

Appendix A

Inner-loop analysis

Consider a single sender, maintaining an amount of $w(t)$ in-flight data at time t, and a sending rate $r(t)$. The data is transmitted over a path with a single bottleneck link of capacity c, and a roundtrip propagation delay τ. The queue that serves the bottleneck link is called the *bottleneck queue*, and is is shared with cross traffic with arrival rate $x(t)$.

The propagation delay can be divided into forward delay τ_{f} (the time it takes for a transmitted packet to arrive to the bottleneck queue) and backward delay τ_{b} (the time it takes for a packet transmitted on the bottleneck link to arrive to the receiver, and for the corresponding ACK to travel back to the sender).

Consider the transmission at some time t_0, and put $t_1 = t_0 + \tau + q(t_0 + \tau_{\mathrm{f}})/c$. Then the ACK for a packet sent at time $t = t_0$ will return to the sender at time $t = t_1$. Furthermore, at time t_1, the ACKs for all packets sent before t_0 have returned, which means that the data in flight at time t_1 is precisely the data sent during the interval $t_0 < t \leq t_1$. Mathematically,

$$w(t_1) = \int_{t_0}^{t_1} r(s)\,\mathrm{d}s$$

Rewrite the left hand side as $w(t_1) = w(t_0) + \int_{t_0}^{t_1} \dot{w}(t)\,\mathrm{d}t$, and move the integral to the right-hand side.

$$w(t_0) = \int_{t_0}^{t_1} \big(r(s) - \dot{w}(t)\big)\,\mathrm{d}t$$

Then, by the mean value theorem, there exists a point ξ in the interval $t_0 \leq \xi \leq t_1$ such that

$$w(t_0) = (t_1 - t_0)\,\big(r(\xi) - \dot{w}(\xi)\big)$$

(FIXME: Notation: Replace ξ by $t_0 + \xi$?) Solving for r, we get

$$r(\xi) = \frac{w(t_0)}{\tau + q(t_0 + \tau_{\mathrm{f}})/c} + \dot{w}(\xi)$$

Here, ξ is unknown and most likely dependent both on t_0 and on the initial conditions. By causality, the time delay for q to influence r is at least τ_b. This implies that $\xi - t_0 \geq \tau$. On the other hand, $\xi \leq t_1$ implies $\xi - t_0 \leq \tau + q(t_0 + \tau_f)/c$, so we get the range

$$\tau \leq \xi - t_0 \leq \tau + q(t_0 + \tau_f)/c$$

To make the model practical, we need to approximate the delay with a constant, $\xi - t_0 = \Delta$. If we require Δ to satisfy the above inequalities, also at times when $q = 0$, $\Delta = \tau$ is the only possible choice.

This leads us to the approximation $\xi \approx t_0 + \tau$. The resulting equation is

$$r(t) \approx \frac{w(t-\tau)}{\tau + q(t-\tau_b)/c} + \dot{w}(t)$$

The time delays in the real system include queueing delays, and are hence state-dependent. Any choice of constant delays in the model is arbitrary to some degree. The choice of fixed delays in the model is a compromise between simplicity and accuracy, with causality requirements taken into account.

The delay in the denominator is clearly motivated by causality; τ_b is the signalling delay between the queue and the sender. The delay in the numerator may at first seem unintuitive; why should there be a signalling delay from the window size, a direct input to the sender's transmission control, to the resulting rate? It can be understood if we interpret the first term as the rate of received ACKs. This rate depends on the window size at the time the corresponding data was sent, and the queue size at the time the data arrived to the queue. The second term represents additional data that is sent or omitted immediately when w is changed.

If we substitute the rate expression into the queue dynamics, $\dot{q}(t) = r(t - \tau_f) + x(t) - c$, we get

$$\dot{q}(t) = \frac{w(t-\tau-\tau_f)}{\tau + q(t-\tau)/c} + \dot{w}(t-\tau_f) + x(t) - c$$

We call this model the *joint link model*.

(FIXME: Refer to stability results here? Probably too complicated to include.)

Appendix B

Information contents of mark bits

This appendix tries to link the information contents of mark bits as used in ECN as well as in the new congestion control mechanism described in Chapter 4. We consider a flow with an average window size of n packets, and a packet mark probability p. For AIMD, the mark probability is $2/n^2$ (derived from the TCP square-root formula,

$$\text{rate} = \frac{\text{packet size}}{\text{roundtrip time}} \sqrt{\frac{2(1-p)}{p}} \tag{B.1}$$

For the proposed congestion control, the mark probability is $1/n$.

This analysis is related to the research on control under information constraints. This field tries to link together fundamental control properties, such as the Bode sensitivity integral, information theoretic properties, such as channel capacity and mutual information, and estimation theory notions such as the Fischer information. I admit I will have to resort to some hand-waving; making such links in the presence of feedback is challenging.

What is the information contents in the feedback bits? Consider a mark bit x which is set by the network, with probability p for a one and $1-p$ for a zero. The probability p is a function of the network state that we would like to estimate. If p is modelled as a stochastic variable with probability density function $f_p(p)$, then the joint probability distribution for p and x is

$$\begin{cases} f_{xp}(0,p) = f_p(p)\,(1-p) \\ f_{xp}(1,p) = f_p(p)\,p \end{cases}$$

The mutual information between x and p, denoted $I(x,p)$, is defined by

$$\mathrm{I}(x,p) = \sum_{x\in\{0,1\}} \int_0^1 f_{xp}(x,p)\, \lg \frac{f_{xp}(y,p)}{f_y(y) f_p(p)}\, \mathrm{d}p$$

This is the amount of information we get about p (and hence about the queue size) by measuring x.

Proposition 1 *The mutual information can be expressed as*

$$\mathrm{I}(x,p) = h(\mathrm{E}[p]) - \mathrm{E}[h(p)]$$

where $h(p)$ *gives the entropy a binary source with a* fixed *probability* p*,*

$$h(p) = -(p \lg p + (1-p)\lg(1-p))$$

Proof: In the expression for $\mathrm{I}(x,p)$, we need the marginal probability $f_x(x)$. We get

$$f_x(0) = \int_0^1 f_p(p)\,(1-p)\,\mathrm{d}p = 1 - \mathrm{E}[p]$$
$$f_x(1) = \int_0^1 f_p(p)\,p\,\mathrm{d}p = \mathrm{E}[p]$$

Substituting the various probability distribution functions into (B) gives

$$\begin{aligned}
\mathrm{I}(x,p) &= \int_0^1 f_p(p)\,(1-p)\lg\frac{f_p(p)\,(1-p)}{(1-\mathrm{E}[p])\,f_p(p)}\,\mathrm{d}p + \int_0^1 f_p(p)\,p\lg\frac{f_p(p)\,p}{\mathrm{E}[p]\,f_p(p)}\,\mathrm{d}p \\
&= \mathrm{E}\left[(1-p)\lg\frac{1-p}{1-\mathrm{E}[p]} + p\lg\frac{p}{\mathrm{E}[p]}\right] \\
&= \mathrm{E}\left[(1-p)\lg(1-p) + p\lg p\right] - \mathrm{E}[1-p]\lg(1-\mathrm{E}[p]) - \mathrm{E}[p]\lg(\mathrm{E}[p]) \\
&= -\mathrm{E}[h(p)] + h(\mathrm{E}[p])
\end{aligned}$$

□

To compute the mutual information, we thus need to know the distribution of p. When the output x is used for feedback, p will depend on the network state which in turn depends on x. The author is not aware of any tools to compute the distribution of p in this setting. So for now, we have to ignore this feedback. Instead, we treat p as a stochastic variable where the source of the randomness is independent cross-traffic with random arrivals. These cause random fluctuations of the queue size, which translate into random fluctuations of p. The mark bits are then used to get information about the queue fluctuations. For simplicity, let us assume p is uniformly distributed over a range $0 \le p \le M \le 1$.

Proposition 2 *Assume that* p *is uniformly distributed on the interval* $[0, M]$*. Then*

$$\mathrm{I}(x,p) = \frac{M}{2} - \left(1 - \frac{M}{2}\right)\lg\left(1 - \frac{M}{2}\right) - \frac{(1-M)^2}{2M}\lg(1-M) - \frac{1}{2\log 2}$$

In particular, $M = 1$ *gives* $I(x,p) = 1 - 1/(2\log 2) \approx 0.27$*, and* M *small gives* $\mathrm{I}(x,p) = (1/2 - 1/(4\log 2))\,M + O(M^2) \approx 0.14\,M + O(M^2)$.

Proof: The probability distribution for p is

$$f_p(p) = \begin{cases} 1/M & 0 \le p \le M \\ 0 & \text{otherwise} \end{cases}$$

with expected value $E[p] = M/2$. Then

$$\begin{aligned} \mathrm{E}[p \lg p] &= \frac{1}{M \log 2} \int_0^M p \log p \, \mathrm{d}p \\ &= \frac{M}{2} \lg M - \frac{M}{4 \log 2} \\ \mathrm{E}[(1-p) \lg (1-p)] &= \frac{1}{M \log 2} \int_0^M (1-p) \log (1-p) \, \mathrm{d}p \\ &= -\frac{(1-M)^2}{2M} \lg (1-M) + \frac{M-2}{4 \log 2} \end{aligned}$$

where $\log$ is the usual logarithm, to base e. Together, we get

$$\begin{aligned} \mathrm{I}(x,p) &= h(\mathrm{E}[p]) - \mathrm{E}[h(p)] = h(M/2) + \mathrm{E}[p \lg p] + \mathrm{E}[(1-p) \lg (1-p) \\ &= -\frac{M}{2} \lg (M/2) - \left(1 - \frac{M}{2}\right) \lg \left(1 - \frac{M}{2}\right) \\ &\quad + \frac{M}{2} \lg M - \frac{M}{4 \log 2} \\ &\quad - \frac{(1-M)^2}{2M} \lg (1-M) + \frac{M-2}{4 \log 2} \\ &= \frac{M}{2} - \left(1 - \frac{M}{2}\right) \lg \left(1 - \frac{M}{2}\right) - \frac{(1-M)^2}{2M} \lg (1-M) - \frac{1}{2 \log 2} \end{aligned}$$

Putting $M = 1$ gives $I(x,p) = 1/2 - 1/2 \lg(1/2) - 1/(2 \log 2) = 1 - 1/(2 \log 2)$. Taylor expansion around $M = 0$ gives

$$\mathrm{I}(x,p) = \frac{M}{2} + \frac{1}{\log 2} \left\{ -\frac{1}{4} M + \frac{1}{24} M^2 + \frac{1}{48} M^3 + O(M^4) \right\}$$

The first non-zero term is $(1/2 - 1/(4 \log 2))\, M$. □

We can now finally compare how much information we get for different average mark probabilities.

For AIMD, the average p is $2/n^2$, so put $M = 4/n^2$. Then we get $0.56/n$ bits per roundtrip time. I.e., the larger the window size, the less information we get per roundtrip time. For the one-packet decrease rule, and an average p of $1/n$, put $M = 2/n$. We then get 0.28 bits of information per roundtrip time, *independent* of the window size.

Appendix C

Stability of a time-delay system

Consider the time-delay system

$$\dot{x}(t) = -ax(t-1)$$

If $x(t)$ is a scalar, and a is a real number, then it's a basic exercise of the argument variation principle to show that the system is stable if and only if a is in the range $0 < a < \pi/2$. We now extend this to the non-scalar case, where X is a vector and A is a square, real, matrix.

Proposition 3 *Consider the system*

$$\dot{X}(t) = -AX(t-1)$$

where X is a vector and A is a real, square, matrix. Assume that all eigenvalues λ of A lie inside the region

$$0 < |\lambda| < \frac{\pi}{2} - |\operatorname{Arg} \lambda|$$

Then the system is asymptotically stable. ***(FIXME: Explain what this means? Arbitrary (continuous? measurable?) initial condition for*** $-1 < t \leq 0$***, and then*** $|x(t)| \to 0$ ***as*** $t \to \infty$***.)***

Proof: We can assume that A is lower triangular (otherwise, it can be made so by a linear change of variables, which doesn't affect the stability properties). Then the equation for x_1 is an independent subsystem, not affected by the rest of the state variables. If it is stable, then $x_1(t) \to 0$ as $t \to \infty$. The equation for x_2,

$$\dot{x}_2(t) = -\lambda_2 x_2(t-1) + a_{21}x_1(t)$$

can be viewed as a system with $a_{21}x_1(t)$ as input and x_2 as output, and if this system is stable, then also $x_2(t) \to 0$ as $t \to \infty$. The argument can then be repeated inductively, to show that a sufficient (and necessary, although

we don't need that here) condition for stability is that each of the *scalar* differential equations

$$\dot{x}_k(t) = -\lambda_k x_k(t-1)$$

is stable. Let us now consider a single eigenvalue, and drop the k subscript for ease of notation. Let

$$\lambda = re^{i\phi}$$

By assumption, $|\phi| < \pi/2$ and $0 < r < \pi/2 - |\phi|$. Consider the characteristic function, which is

$$f(s) = s + re^{i\phi - s}$$

Let us apply the argument variation principle, and use the standard contour enclosing the right half-plane, where the interesting part is the curve

$$f(i\omega) = r\cos(\phi - \omega) + i[\omega + r\sin(\phi - \omega)]$$

Since $|\phi| < \pi/2$, we have $\operatorname{Re} f(0) > 0$. Consider the crossings with the imaginary axis, which are given by

$$\omega_k = \phi - \pi/2 + k\pi$$
$$f(i\omega_k) = i\left[\phi - \pi/2 + k\pi + (-1)^k r\right]$$

We need to consider both cases $\omega > 0$ and $\omega < 0$ (for $\phi \neq 0$, these are not simply mirror images). For $k > 0$, we get $\omega_k > 0$, and $\operatorname{Im} f(i\omega_k) \geq \pi/2 + \phi - r \geq \pi/2 - |\phi| - r > 0$. And for $k \leq 0$, we get $\omega_k < 0$, and $\operatorname{Im} f(i\omega_k) \leq |\phi| - \pi/2 + r < 0$.

Since all crossings with $\omega > 0$ are in the upper half-plane and all crossings with $\omega < 0$ are in the lower half plane, it follows that the curve does not encircle the origin, and hence the characteristic function has no roots in the left half plane. Since this is valid for each eigenvalue, this concludes the proof. □

We need to apply these proposition to the case of a 2×2 matrix A, so let us consider this special case in some more detail.

Proposition 4 *Let A be a 2×2 matrix. Let $D = \det A$ and $T = \operatorname{tr} A$, and assume that*

$$0 < D < \frac{\pi^2}{4}$$
$$2\sqrt{D}\sin\sqrt{D} < T < \frac{\pi}{2} + \frac{2}{\pi}D$$

Then the system

$$\dot{X}(t) = -AX(t-1)$$

is stable.

Proof: The characteristic polynomial of A is

$$\lambda^2 - T\lambda + D$$

The eigenvalues are the zeros of this polynomial,

$$\lambda = T/2 \pm \sqrt{T^2/4 - D}$$

We will show that these lie in the area defined by the previous proposition. We need to consider the cases of real and complex roots separately. If the roots are real, then since $D > 0$, both roots are real and positive. We get

$$T^2/4 - D < \frac{1}{4}\left(\frac{\pi}{2} + \frac{2}{\pi}D\right)^2 - D = \frac{1}{4}\left(\frac{\pi^2}{4} - \frac{2}{\pi}D\right)^2$$

In the case of real roots, the left hand side is non-negative, and then so is the right hand side, so we can take square roots to to get

$$\lambda \leq T/2 + \sqrt{T^2/4 - D} < \frac{1}{2}\left(\frac{\pi}{2} + \frac{2}{\pi}D\right) + \frac{1}{2}\left(\frac{\pi}{2} - \frac{2}{\pi}D\right) = \frac{\pi}{2}$$

Remains the case of complex roots. We then have $|\lambda| = \sqrt{D}$ and $|\operatorname{Arg}\lambda| = \arccos(T/(2\sqrt{D}))$. We want to show that $|\lambda| < \pi/2 - |\operatorname{Arg}\lambda|$. Rewrite the righthand side,

$$\frac{\pi}{2} - |\operatorname{Arg}\lambda| = \frac{\pi}{2} - \arccos\frac{T}{2\sqrt{D}} = \arcsin\frac{T}{2\sqrt{D}} > \sqrt{D} = |\lambda|$$

where the inequality follows from the assumption $T/(2\sqrt{D}) > \sin\sqrt{D}$ by applying the increasing $\arcsin$ function to both sides. This concludes the proof. □

Corollary 5 *Let A be a 2×2 matrix. Let $D = \det A$ and $T = \operatorname{tr} A$, and assume that*

$$0 < D < \frac{\pi^2}{4}$$
$$2D < T < \frac{\pi}{2} + \frac{2}{\pi}D$$

Then the system

$$\dot{X}(t) = -AX(t-1)$$

is stable.

Proof: Follows from the inequality $\sin\sqrt{D} \leq \sqrt{D}$, applied to the previous proposition. □

Appendix D

Outer-loop analysis

This appendix analyzes the congestion control mechanism described in Chapter 4. We consider the simplest topology, with a single flow and a single bottleneck, but allow for some cross traffic at the bottleneck.

(FIXME: Better structure.)

D.1 Model

(FIXME: Define notation)

The inner-loop is described by the differential equation

$$\dot{q}(t) = \frac{w(t-\tau)}{\tau + q(t-\tau)/c} + \dot{w}(t) - \gamma c$$

This is the joint link model described in Appendix A. We combine this with the equation for the window adjustments,

$$\dot{w}(t) = \frac{1}{\tau + q(t-\tau)/c}\left(m - \frac{q(t-\tau)}{q_0 + q(t-\tau)}\, w(t-\tau)\right)$$

Here, the first term, $m/(\tau+q(t-\tau)/c)$, is the additive increase of one packet per roundtrip time, and the second term is the additive decrease, which is the product of the current ACK-rate, and the marking probability at the time the currently ACKed packet left the queue, $q(t-\tau)/(q_0 + q(t-\tau))$.

D.2 Equlibrium

Let q^* and w^* denote the equilibrium values, then the equilibrium equations are

$$\gamma c = \frac{w^*}{\tau + q^*/c}$$
$$m = \frac{q^* w^*}{q_0 + q^*}$$

Also let τ^* denote the corresponding roundtrip time, $\tau^* = \tau + q^*/c$. The equilibrium equations can be solved explicitly (which also shows that there is a unique positive solution):

$$q^* = -\frac{\gamma c\tau - m}{2\gamma} + \frac{1}{\gamma}\sqrt{\frac{(\gamma c\tau - m)^2}{4} + \gamma q_0 m}$$
$$w^* = \frac{\gamma c\tau + m}{2} + \sqrt{\frac{(\gamma c\tau - m)^2}{4} + \gamma q_0 m}$$

In particular, for large q_0, both q^* and w^* grow as $\sqrt{q_0}$.

D.3 Stability

Let $\widetilde{q} = q(t) - q^*$ and $\widetilde{w}(t) = w(t) - w^*$ denote the deviation from the equilibrium, and scale time by defining

$$u(t) = \begin{bmatrix} \widetilde{q}(\tau\, t) \\ \widetilde{w}(\tau\, t) \end{bmatrix}$$

Then, after linearization, we get

$$\dot{u}(t) = -Au(t-1)$$

with

$$A = \frac{\tau}{\tau^*}\begin{bmatrix} \dfrac{w^* q_0}{(q_0+q^*)^2} + \gamma & -\dfrac{q_0}{q_0+q^*} \\ \dfrac{w^* q_0}{(q_0+q^*)^2} & \dfrac{q^*}{q_0+q^*} \end{bmatrix}$$

Substituting $w^* = \gamma c\tau^*$ gives

$$A = \frac{\tau}{\tau^*}\begin{bmatrix} \dfrac{\gamma c\tau^* q_0}{(q_0+q^*)^2} + \gamma & -\dfrac{q_0}{q_0+q^*} \\ \dfrac{\gamma c\tau^* q_0}{(q_0+q^*)^2} & \dfrac{q^*}{q_0+q^*} \end{bmatrix}$$

Let $D = \det A$ and $T = \operatorname{tr} A$, then

$$T = \frac{\tau}{\tau^*}\left(\gamma + \frac{q^*}{q_0+q^*}\right) + \gamma\frac{c\tau q_0}{(q_0+q^*)^2}$$
$$D = \gamma\frac{\tau}{\tau^*}\left(\frac{\tau}{\tau^*}\frac{q^*}{q_0+q^*} + \frac{c\tau q_0}{(q_0+q^*)^2}\right)$$

Assume that $q_0 \geq \alpha c\tau$. Then we can bound

$$\frac{c\tau q_0}{(q_0+q^*)^2} < \frac{1}{\alpha}$$

Furthermore, since the function $f(x) = x/[(1+x)(\alpha+x)]$ has the maximum value $1/(1+\sqrt{\alpha})^2)$ we get

$$\begin{aligned}\frac{\tau}{\tau^*}\frac{q^*}{q_0+q^*} &\leq \frac{q^*/(c\tau)}{(1+q^*/(c\tau))(\alpha+q^*/(c\tau))} \\ &\leq \frac{1}{(1+\sqrt{\alpha})^2} < \frac{1}{\alpha}\end{aligned}$$

It follows that

$$\begin{aligned}T &< \gamma + \frac{1+\gamma}{\alpha} < 1 + \frac{2}{\alpha} \\ D &< \frac{2\gamma}{\alpha} < \frac{2}{\alpha}\end{aligned}$$

So we can make both T and D small by selecting a large α, i.e., a large value for the parameter q_0. To apply Corollary 5, we also need $2D < T$. We have

$$\begin{aligned}T - 2D &= \gamma\frac{\tau}{\tau^*} + \left(1 - 2\gamma\frac{\tau}{\tau^*}\right)\frac{\tau}{\tau^*}\frac{q^*}{q_0+q^*} + \gamma\left(1-2\frac{\tau}{\tau^*}\right)\frac{c\tau q_0}{(q_0+q^*)^2} \\ &\geq \gamma\frac{\tau}{\tau^*} + \gamma\left(1-2\frac{\tau}{\tau^*}\right)\left(\frac{\tau}{\tau^*}\frac{q^*}{q_0+q^*} + \frac{c\tau q_0}{(q_0+q^*)^2}\right)\end{aligned}$$

If $\tau^* \geq 2\tau$, this is clearly positive. So assume that $\tau^* < 2\tau$ (which should be the usual case, if parameters are choosen for a small queue size). We then have

$$\begin{aligned}\frac{T-2D}{\gamma} &\geq \frac{\tau}{\tau^*} - \left(2\frac{\tau}{\tau^*} - 1\right)\left(\frac{\tau}{\tau^*}\frac{q^*}{q_0+q^*} + \frac{c\tau q_0}{(q_0+q^*)^2}\right) \\ &\geq \frac{\tau}{\tau^*} - \left(2\frac{\tau}{\tau^*} - 1\right)\frac{2}{\alpha} = \frac{\tau}{\tau^*}\left(1 - \frac{4}{\alpha}\right) + \frac{2}{\alpha}\end{aligned}$$

This is positive provided $\alpha \geq 4$. We can now formulate a stability condition.

Proposition 6 *Consider the system*

$$\begin{aligned}\dot{q}(t) &= \frac{w(t-\tau)}{\tau + q(t-\tau)/c} + \dot{w}(t) - \gamma c \\ \dot{w}(t) &= \frac{1}{\tau + q(t-\tau)/c}\left(m - \frac{q(t-\tau)}{q_0 + q(t-\tau)}\,w(t-\tau)\right)\end{aligned}$$

modelling the dynamics of the proposed congestion control mechanism. Assume that $q_0 \geq 4c\tau$. Then this system is locally stable.

Proof: Above, we have derived the linearized system,

$$\dot{u}(t) = -Au(t-1)$$

characterized by a 2×2 matrix A with trace T and determinant D. If $q_0 \geq 4c\tau$, we have the bounds

$$\begin{aligned} T &< \frac{3}{2} < \frac{\pi}{2} \\ D &< \frac{1}{2} < \frac{\pi^2}{4} \\ T &> 2D \end{aligned}$$

Then the hypotheses of Corollary 5 are satisfied. The linearized system is asymptotically stable, and then the original non-linear system is locally stable. □

The above stability condition is a bit conservative. If we consider the case $q_0 = c\tau$, we get

$$\begin{aligned} q^* &= \frac{m}{\gamma} \\ w^* &= \gamma c\tau + m \\ \tau^* &= \tau + \frac{m}{\gamma c} \end{aligned}$$

Put $\beta = \tau/\tau^* = \gamma c\tau/(\gamma c\tau + m) < 1$. Then

$$\begin{aligned} T &= \beta(\gamma + 1 - \beta) + \gamma\beta^2 \\ D &= \gamma\beta^2 \end{aligned}$$

It follows that $D < 1$, and

$$\begin{aligned} T - \frac{2}{\pi}D &< 2\beta - \frac{2}{\pi}\beta^2 < 2 - \frac{2}{\pi} < \frac{\pi}{2} \\ T - 2D &= \beta(\gamma + 1 - \beta) - \gamma\beta^2 = \beta(\gamma + 1)(1 - \beta) > 0 \end{aligned}$$

so also in this case, Corollary 5 can be applied. It may be possible, with a bit more effort, to prove stability under the condition $q_0 \geq c\tau$.

D.4 Simulation

To get a better understanding of the dynamics, and to see the influence of the system parameter γ and the design parameter q_0, we can solve the differential equations numerically. As base values, consider a 10 Mbit/s link, with a roundtrip propagation delay of 50 ms, and with 30% non-responsive cross-traffic (i.e. $\gamma = 0.7$). The packet size is 1500 bytes, and we set the control parameter q_0 to 150 packets. For comparison, the pipe size $c\tau$ is 42 packets. For all the simulations, the initial conditions are $q(t) = 0$ and $w(t) = w^*$ for $t \leq 0$.

We first vary γ. We have seen in Section 3.3 that the time constant of the inner-loop grow large when γ is decreased. In Figure D.1, we see that

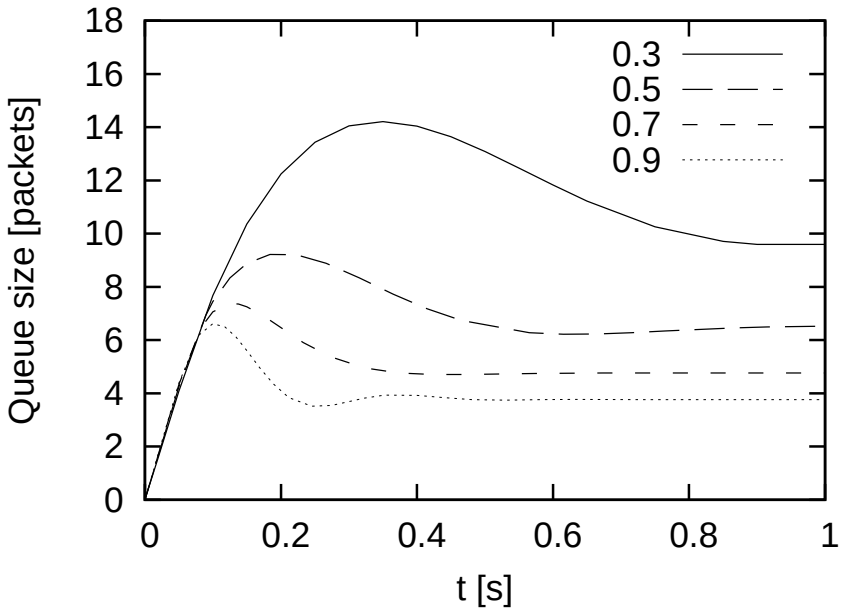

Figure D.1: The evolution of the queue size, for $\gamma = 0.3$, 0.5, 0.7, and 0.9.

also the dynamics of the closed loop window control gets slower as γ is made smaller, i.e., when the amount of cross-traffic is increased. We can also see that when γ is decreased, the equilibrium queue grows (window size is not shown in this figure, but the equlibrium window size shrinks with decreased γ, which makes sense since the "effective pipe size" $\gamma c\tau$ also shrinks).

Next, consider the influence of the propagation delay τ. In Figure D.2 we see the dynamics when the delay is increased from 50 ms up to 300 ms. From the curves corresponding to a propagation delay of 50 ms and 100 ms, the system is stable with a convergence time of about 5 times the propagation delay. For 200 ms delay, the system is still stable, but with much slower convergence. Finally, for 300 ms delay, the system approaches a limit cycle where the queue oscillates between zero and approximately two packets. Since the cycle includes an interval when the queue is empty, there is also a slight underutilization of the link in the scenario with 300 ms delay.

For all the values of τ considered here, stable or unstable, the size of transients and oscillations in the queue size are at most a couple of packets. For τ even larger than 300 ms, both the oscillations and the utilization would get smaller, and q_0 must be increased to recover full utilization.

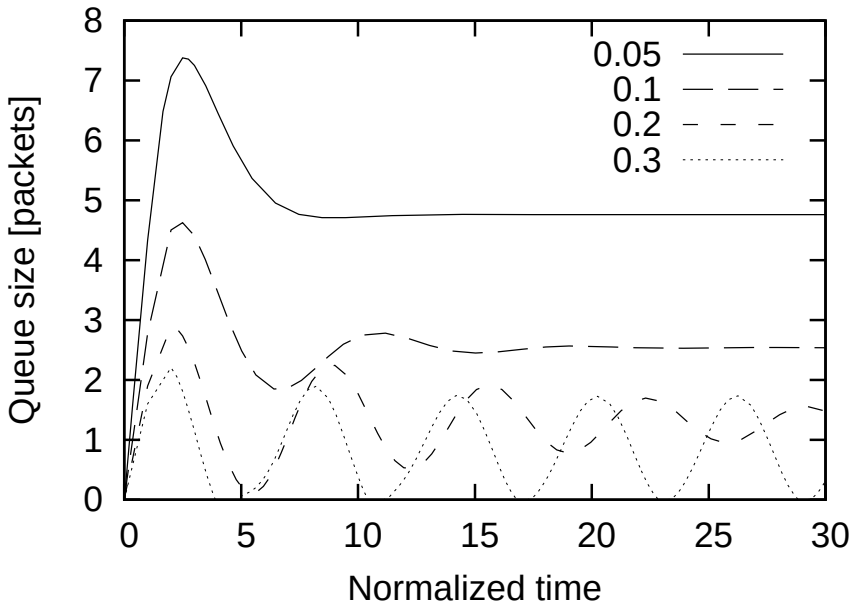

Figure D.2: The evolution of the queue size, for $\tau = 0.05$, 0.1, 0.2, and 0.3. To aid comparison, the horizontal axis is in units of τ.